"This Hangman game implemented in Python demonstrates a clear and well-structured approach to coding a classic word-guessing game. The program showcases several virtues that make it not only functional but also easy to understand and extend.

Overall, this Hangman game implementation is a stellar example of clean, well-documented, and user-friendly Python programming. It effectively combines modular design, robust input handling, and engaging gameplay mechanics to create an enjoyable and maintainable application.

- **Modular Design**: The code is divided into distinct functions, each handling specific tasks. This makes the program easier to read, maintain, and debug. Functions like set_word(), get_player_input(), update_players_letters(), and play_hangman() each have well-defined responsibilities, enhancing the modularity of the code.
- **Clear Documentation**: The use of docstrings for each function provides clear explanations of what each function does, its arguments, and its return values. This is invaluable for anyone who needs to understand or modify the code later.
- **User-Friendly Interaction**: The get_player_input() function is designed to guide the player through the game by displaying the current state of guesses and prompting for input. This ensures an engaging and interactive user experience.
- **Robust Input Handling**: The program checks for various types of input, such as invalid characters and repeated guesses, and handles them gracefully. This prevents the game from crashing due to unexpected input and provides feedback to the player.
- **Game Logic Implementation**: The play_hangman() function encapsulates the main game logic, including tracking incorrect guesses, updating the display based on correct guesses, and determining win/loss conditions. The function is comprehensive and handles all aspects of gameplay effectively.
- **Testing Mode**: The TESTING flag allows developers to hardcode a secret word for testing purposes. This is a practical feature that facilitates debugging and testing without the need to rely on random word selection.
- **Random Word Selection**: The use of the randint() function to select a random word from a predefined list adds an element of unpredictability to the game, enhancing replayability.
- **Visual Feedback**: The integration of hangman_art.stages provides visual feedback for incorrect guesses, making the game more engaging and helping players visualize their progress.
- **Comprehensive Feedback**: Throughout the game, players are informed about their correct and incorrect guesses, the remaining number of attempts, and the current state of the word. This continuous feedback keeps players informed and engaged.
- **Clean Main Function**: The main() function acts as an entry point to the game, setting up necessary variables, starting the game, and printing the outcome. This clear separation of setup, execution, and conclusion enhances readability and organization."

--chatGPT 3.5, OpenAI

“This Hangman program is a well-structured and engaging implementation of the classic word guessing game. Here's why it shines:

Overall, this program demonstrates a strong understanding of core programming concepts and effectively translates the Hangman game into a fun and interactive experience.

- **Clear Functionality**: The code effectively captures the essence of Hangman. Players try to guess a secret word letter by letter, with visual feedback on their progress and remaining guesses.
- **Modular Design**: The code is broken down into well-defined functions, making it easier to understand, maintain, and potentially extend with new features.
- **Meaningful Variable Names**: Descriptive variable names like secret_word and bad_guesses_remaining enhance readability.
- **User-Friendly Input**: The program prompts the user for input clearly, handles invalid entries gracefully, and even allows exiting the game mid-play.
- **Informative Feedback**: The program provides helpful messages to the player throughout the game, indicating correct guesses, used letters, and the remaining number of guesses.
- **Testing Mode**: The inclusion of a testing mode (TESTING flag) allows for easier debugging and verification of game logic.”

--Gemini, Google AI

```
#hangman.py  by  msbridges 7/2/2024


import os
from typing import List, Union
from random import randint
import hangman_words
import hangman_art

TESTING = True  # Set to True in order to hardcode a secret word in main()

def set_word() -> tuple[str, list[str], list[str]]:
    """
    Sets up the game by fetching a random word from the hangman_words file.
    initializes the list variables needed to run the game--one to contain
    the word's letter, and the other to contain the players guesses.

    Returns:
        tuple: (secret_word: str, letters_of_word: list[str], players_letters_of_word: list[str])
    """

    # Fetch secret word, letters of word list
    r = randint(1, 214)
    secret_word = hangman_words.word_list[r]
    secret_word_length = len(secret_word)

    # Create letters of word list
    letters_of_word = [char for char in secret_word if char.isalpha()]

    # Create players letters of word list - visual aid
    players_letters_of_word = ['_'] * secret_word_length

    return secret_word, letters_of_word, players_letters_of_word
```

```python
def get_player_input(players_letters_of_word: List[str]) -> str:
    """
    This function prompts the player to enter a letter he/she guesses
    might be in the secret word. The player's correct and incorrect
    guesses so far are displayed for consideration.

    Args:
        players_letters_of_word(List[chr]): the player's correctly
        guessed letters
    Returns:
        Either:
        1) player_input(char): the lower case of the valid single
            alphabetic char input by the player.
        2) player_input(string): the string 'quit' to exit the game.
        3) player_input(string): the string 'invalid' to indicate a
            non-single alphabetic char was entered.
    """

    # Prompt player for a letter, display to player state of guesses
    players_letters_of_word_str = ' '.join(players_letters_of_word)
    print(f'get_player_input(): So far you have guessed these letters: {players_letters_of_word_str}\n')
    player_input = input('get_player_input(): Guess a single letter or enter \'quit\': ')

    player_input_length = len(player_input)
    if player_input_length == 1 and player_input.isalpha():
        return player_input.lower()
    elif player_input.lower() == 'quit':
        return 'quit'
    else:
        return 'invalid'

def update_players_letters(occurrence_indices: List[int], letters_of_word: List[str], players_letters_of_word: List[str]) -> List[str]:
    """
    Write letter(s) to players_letters_of_word

    Args:
        occurrence_indices(List[int]): position(s) of letter guessed by player found in the word
        letters_of_word(List[str]): letters of the word
        players_letters_of_word(List[str]): correctly guessed letters so far

    Returns:
        players_letters_of_word(List[str]): updated list of correctly guessed letters
    """

    for index in occurrence_indices:
        players_letters_of_word[index] = letters_of_word[index]

    return players_letters_of_word

def play_hangman(letters_of_word: List[str], players_letters_of_word: List[str]) -> Union[str, bool]:
    """
    This function implements the Hangman game. The object of the game is
    for the player to guess a secret word letter by letter before a
    drawing of a stickman hanging from a gallows can be completed.
    Each time the player guesses an incorrect letter, the drawing
    progresses. After six incorrect guesses, the drawing is complete
    and the game is over. But if the player correctly guesses all of
    the letters in the word before entering six incorrect guesses,
    the player wins. The game only accepts single letters as valid
    entries. Invalid entries do not count as incorrect guesses. If
    the word has multiple occurrences of a letter, all are counted.
```

```
For example, if the word is 'hello' and the letter 'l' is guessed,
both 'l's are counted. The player is shown both the correct and
incorrect guesses during the game. The player can stop the game
by entering 'quit'.

Args:
    letters_of_word: List[str]: letters of the word
    players_letters_of_word: List[str]: correctly guessed letters so far
Returns:
    Union[str, bool]: Either 'quit' (string) or True/False (bool) indicating win/loss
"""

bad_guesses_remaining = 6
letters_used_str = ''

while (bad_guesses_remaining > 0):
    occurrence_indices = []

    # Get player's letter
    player_input = get_player_input(players_letters_of_word)

    if player_input == 'quit':
        return 'quit'

    if player_input == 'invalid':
        print(f'play_hangman(): The entry was {player_input}.')
        continue

    letter = player_input  # For readability

    # Is letter used?
    if letter in letters_used_str:
        print(f'play_hangman(): The letter {letter} has alread been used, try another.')
        continue

    # Is letter in word?
    if letter in  letters_of_word:
        print(f'play_hangman(): Good guess!')

        # Yes, make letter occurrences index list
        occurrence_indices = [i for i, char in enumerate(letters_of_word) if char == letter]
        print(f'play_hangman(): There is {len(occurrence_indices)} occurrence(s) of \'{letter}\'.')

        # Update player's correctly guessed letters list
        players_letters_of_word = update_players_letters(occurrence_indices, letters_of_word, players_letters_of_word)

        # Did player win?
        if letters_of_word == players_letters_of_word:
            return True

    # No, draw next stage of hanging man
    else:
        print(f'play_hangman(): Bad guess! Enter a single letter or \'quit\'\n')
        letters_used_str = letters_used_str + ' ' + letter
        print(hangman_art.stages[bad_guesses_remaining - 1])
        bad_guesses_remaining -= 1
        print(f'\nplay_hangman(): Used letters: {letters_used_str}')

        # Did player loose?
        if  bad_guesses_remaining == 0:
            return False
```

```
def main():
    """
    Obtains secret word, launches the game, prints the outcome.
    Provides for a testing mode where key variables may be initialized manually.

    """
    os.system('cls')
    print(f'main(): Welcome! Lets play Hangman.\n')

    # Hardcode variables for testing here
    if TESTING == True:
        secret_word = 'hello'
        letters_of_word = ['h', 'e', 'l', 'l', 'o']
        players_letters_of_word = ['_', '_', '_', '_', '_']
        print(f'main(): Psst.. the secret word is: {secret_word}\n')
    else:
        # Fetch secret word
        secret_word, letters_of_word, players_letters_of_word = set_word()

    # Begin game
    winner = play_hangman(letters_of_word, players_letters_of_word)

    # Print outcome
    if winner == True:
        print(f'main(): Congrats! You won. The word was \'{secret_word}\'.\n')
    elif winner == False:
        print(f'main(): Sorry! You lost. The word was \'{secret_word}\'.\n\n')
    elif winner == 'quit':
        print(f'main(): Exiting Hangman. Thanks for playing!\n.')

if __name__ == '__main__':
    main()
```